B2

PAT-NO: JP02001282547A

DOCUMENT-IDENTIFIER: JP 2001282547 A

TITLE: FUNCTION ALLOCATION/OPTIMIZATION DEVICE TO INSTRUCTION CACHE, FUNCTION ALLOCATION/OPTIMIZATION METHOD AND RECORDING MEDIUM RECORDING FUNCTION ALLOCATION/ OPTIMIZATION PROCEDURE

PUBN-DATE: October 12, 2001

INVENTOR-INFORMATION:

NAME	COUNTRY
TANAKA, EISHIN	N/A

ASSIGNEE-INFORMATION:

NAME	COUNTRY
NEC MICROSYSTEMS LTD	N/A

APPL-NO: JP2000089382

APPL-DATE: March 28, 2000

INT-CL (IPC): G06F009/45, G06F011/34 , G06F012/08

ABSTRACT:

PROBLEM TO BE SOLVED: To reduce a cache conflict among a plurality of functions and to improve the performing speed of an application program.

SOLUTION: A function call information output part 1, a function basic block shift information output part 2 outputting function basic block shift information 112 where the combination of ID of the function and the order of

basic blocks is arranged at every shift of the function corresponding to a function call and a function memory arrangement/optimization part 3 which refers to function call combination information 111, generates call number exchange information 113 constituted of function call number exchange data where the number of shift times is exchanged, temporarily arranges the function by referring to call number exchange information 113, detects the number of conflict times by referring to function basic block shift information 112, decides the memory arrangement of the function to data whose number of conflict times is the smallest among call number exchange data and outputs a corresponding function memory arrangement result 4 are installed.

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号
特開2001－282547
（P2001－282547A）
(43)公開日　平成13年10月12日(2001.10.12)

(51)Int.Cl.⁷	識別記号	ＦＩ		テーマコード（参考）
Ｇ０６Ｆ　9/45		Ｇ０６Ｆ　11/34	Ｓ	５Ｂ００５
11/34		12/08	Ｗ	５Ｂ０４２
12/08		9/44	３２２Ｈ	５Ｂ０８１

審査請求　有　　請求項の数７　ＯＬ　（全 16 頁）

(21)出願番号　特願2000－89382(P2000－89382)

(22)出願日　平成12年３月28日(2000.3.28)

(71)出願人　000232036
エヌイーシーマイクロシステム株式会社
神奈川県川崎市中原区小杉町１丁目403番53

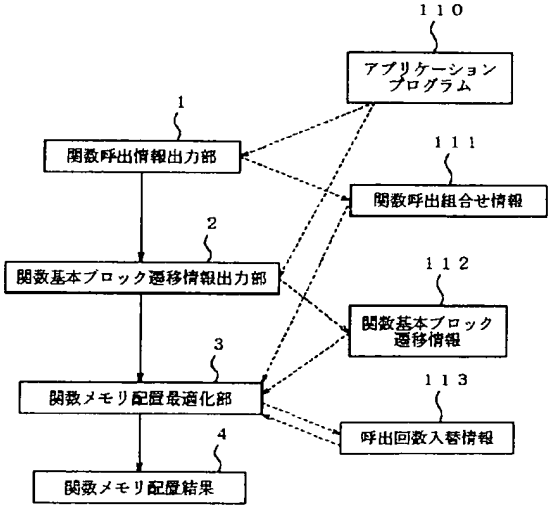
(72)発明者　田中　英信
神奈川県川崎市中原区小杉町一丁目403番53　日本電気アイシーマイコンシステム株式会社内

(74)代理人　100082935
弁理士　京本　直樹　（外２名）

Ｆターム(参考)　5B005 JJ13 KK12 LL01 MM02 QQ04
TT00 VV02 VV24
5B042 GA02 HH20 MC25
5B081 CC27

(54)【発明の名称】　命令キャッシュへの関数割付最適化装置、関数割付最適化方法及び関数割付最適化手順を記録した記録媒体

(57)【要約】
【課題】複数の関数間のキャッシュコンフリクトを削減し、アプリケーションプログラムの実行スピードの向上を図る。
【解決手段】関数呼出情報出力部１と、関数呼出対応の関数の遷移毎に該関数のＩＤと基本ブロックの順番の組合せを並べた関数基本ブロック遷移情報１１２に出力する関数基本ブロック遷移情報出力部２と、関数呼出組合せ情報１１１を参照し遷移回数を入替えた関数呼出回数入替データから成る呼出回数入替情報１１３を生成し、呼出回数入替情報１１３を参照して関数を仮配置した後、関数基本ブロック遷移情報１１２を参照しコンフリクト回数を検出し、呼出回数入替データの中でコンフリクト回数の最少のものに関数のメモリ配置を決定し対応の関数メモリ配置結果４を出力する関数メモリ配置最適化部３とを備える。



1

【特許請求の範囲】

【請求項1】　命令キャッシュを搭載したマイクロプロセッサシステム用の所定のアプリケーションプログラムを入力し前記命令キャッシュに関しキャッシュコンフリクトの回数を確率的に低減するように関数の呼出回数情報に基づいて関数のメモリ配置を最適化する命令キャッシュへの関数割付最適化装置において、

前記アプリケーションプログラムを入力しプロファイルによる関数呼出時に呼出元及び呼出先の各関数とその呼出回数とを関数呼出組合せとして関数呼出組合せ情報に出力する関数呼出情報出力部と、

前記アプリケーションプログラムを入力し前記プロファイルによる関数呼出に応じた関数の遷移に対して該関数のＩＤ及び該関数の基本ブロックの順番の組合せを関数遷移毎に並べた関数基本ブロック遷移情報に出力する関数基本ブロック遷移情報出力部と、

前記関数呼出組合せ情報を参照し前記関数呼出組合せ相互間で前記関数遷移の回数である遷移回数を入替えた関数呼出回数入替データから成る呼出回数入替情報を生成し、次に生成した前記呼出回数入替情報を参照して関数をメモリ空間上のアドレスに仮配置した後、前記関数基本ブロック遷移情報を参照してキャッシュコンフリクトの回数を検出し、前記呼出回数入替データの中で、前記キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定し対応する関数メモリ配置結果を出力する関数メモリ配置最適化部とを備えることを特徴とする命令キャッシュへの関数割付最適化装置。

【請求項2】　前記関数呼出組合せ情報及び前記呼出回数入替情報の各々が、関数の呼出元の関数名を記述した呼出元欄と、

前記関数の呼出先の関数名を記述した呼出先欄と、

前記関数の呼出回数を設定する呼出回数欄とをそれぞれ有することを特徴とする請求項1記載の命令キャッシュへの関数割付最適化装置。

【請求項3】　命令キャッシュを搭載したマイクロプロセッサシステム用のアプリケーションプログラムを入力し前記命令キャッシュに関しキャッシュコンフリクトの回数を確率的に低減するように関数の呼出回数情報に基づいて関数のメモリ配置を最適化する命令キャッシュへの関数割付最適化方法において、

前記アプリケーションプログラムを入力し前記プロファイルによる関数呼出時に呼出元及び呼出先の各関数とその呼出回数とを関数呼出組合せとして関数呼出組合せ情報を生成し、

前記アプリケーションプログラムを入力し前記プロファイルにより得られた前記関数の基本ブロック単位の実行に関する関数基本ブロック遷移情報を生成した後、

前記関数呼出組合せ情報を参照し前記関数呼出組合せ相互間で前記関数遷移の回数である遷移回数を入替えた関数呼出回数入替データから成る呼出回数入替情報を生成し、前記関数基本ブロック遷移情報を参照して各関数のキャッシュコンフリクトの回数を検出し、前記呼出回数入替データの中で前記キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定する関数メモリ配置最適化工程を有することを特徴とする命令キャッシュへの関数割付最適化方法。

2

【請求項4】　前記関数メモリ配置最適化工程が、前記関数呼出組合せ情報を参照して前記呼出回数入替情報を生成する呼出回数入替ステップと、

生成した前記呼出回数入替情報を参照して関数をメモリ空間上のアドレスに仮配置する関数メモリ仮配置ステップと、

前記関数基本ブロック遷移情報を参照して仮配置した各関数のキャッシュコンフリクト回数を検出するキャッシュコンフリクト回数算出処理ステップと、

前記関数呼出回数入替データの中で前記キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定する関数メモリ配置ステップとを有することを特徴とする請求項3記載の命令キャッシュへの関数割付最適化方法。

【請求項5】　前記呼出回数入替ステップが、前記関数呼出組合せ情報を参照して呼出回数の入替対象とする前記関数呼出組合せの数である引数を読み込み第1の変数を設定する第1のステップと、

前記第1の変数が0か否かを判定する第2のステップと、

前記第2のステップで前記第1の変数が0の場合現在の内容の呼出回数入替情報を出力する第3のステップと、

前記第2のステップで前記第1の変数が0以外の場合呼出回数入替処理を行い前記第1の変数－1に対応する引数を設定して再帰呼出を行う第4のステップと、

第2の変数に0を設定する第5のステップと、

前記第2の変数が前記第1の変数－1より小さいか否かの判定を行い否の場合は処理を終了する第6のステップと、

前記第6のステップで諾の場合前記第2の変数と前記第1の変数－1である第1のインッデクスの各々の前記呼出回数を交換する第7のステップと、

前記引数を1デクリメントして前記呼出回数入替処理を行い前記第1の変数－1に対応する引数を設定して再帰呼出を行う第8のステップと、

前記第2の変数である第2のインデックスと前記第1の変数－1の各々の前記呼出回数を交換する第9のステップと、

前記第2の変数を1インクリメントし前記第6のステップ以降を反復する第10のステップとを有することを特徴とする請求項4記載の命令キャッシュへの関数割付最適化方法。

【請求項6】　前記キャッシュコンフリクト回数算出処理ステップが、キャッシュコンフリクト回数をカウント

する変数を0に初期化する第1のステップと、
前記関数基本ブロック遷移情報を順次読み込み、関数IDと基本ブロックの順番情報であるID順番情報を求める第2のステップと、
前記関数基本ブロック遷移情報は終了したかを判定し諾の場合は処理を終了する第3のステップと、
前記第3のステップで否の場合は前記ID順番情報のキャッシュ上の配置を求める第4のステップと、
先頭の前記関数基本ブロック遷移情報かを判定し諾の場合は前記第2のステップに戻る第5のステップと、
前記第5のステップで否の場合以前のブロックとアドレス上の重なりがあるかを判定し否の場合は前記第2のステップに戻る第6のステップと、
前記第6のステップで諾の場合は前記キャッシュコンフリクト回数をカウントする変数を1インクリメントし前記第2のステップに戻る第7のステップとを有することを特徴とする請求項4記載の命令キャッシュへの関数割付最適化方法。

【請求項7】　命令キャッシュを搭載したマイクロプロセッサシステム用のアプリケーションプログラムを入力し前記命令キャッシュに関しキャッシュコンフリクトの回数を確率的に低減するように関数の呼出回数情報に基づいて関数のメモリ配置を最適化する命令キャッシュへの関数割付最適化手順を記録した記録媒体において、
前記アプリケーションプログラムを入力し前記プロファイルによる関数呼出時に呼出元及び呼出先の各関数とその呼出回数とを関数呼出組合せとして関数呼出組合せ情報を生成する手順と、
前記アプリケーションプログラムを入力し前記プロファイルにより得られた前記関数の基本ブロック単位の実行に関する関数基本ブロック遷移情報を生成する手順と、
前記関数呼出組合せ情報を参照し前記関数呼出組合せ相互間で前記関数遷移の回数である遷移回数を入替えた関数呼出回数入替データから成る呼出回数入替情報を生成し、前記関数基本ブロック遷移情報を参照して各関数のキャッシュコンフリクトの回数を検出し、前記呼出回数入替データの中で前記キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定する関数メモリ配置最適化手順とを実行させるプログラムを記録したことを特徴とする命令キャッシュへの関数割付最適化手順を記録した記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は命令キャッシュへの関数割付最適化装置、関数割付最適化方法及び関数割付最適化手順を記録した記録媒体に関し、特にキャッシュを搭載したマイクロプロセッサシステムにおける命令キャッシュへの関数割付最適化装置、関数割付最適化方法及び関数割付最適化手順を記録した記録媒体に関する。

【0002】

【従来の技術】この種のマイクロプロセッサシステムの処理速度においては、CPU速度に加えて主記憶である外部メモリに対するアクセス、すなわち、メモリアクセスの処理速度（以下メモリアクセス速度）が大きく影響する。しかしながら、最近のCPU速度の著しい向上に対して、メモリアクセス速度の向上はそれほど大きくなく、CPU速度とメモリアクセス速度との差は年々開く一方である。例えば、DRAMのアクセス時間の向上率は年率7%程度であるのに対し、CPU速度の向上率は50～100%というデータもある（ダビド・エー・パターソン、ジョンエル・ヘネシー（David A. Patterson、John. L. Hennessy）著、成田光彰訳、「コンピュータの構成と設計」、日経BP社、1996年4月19日）。

【0003】従って、システム処理速度の向上のためには、キャッシュの有効利用が、非常に重要な問題となってきている。なお、キャッシュとは、公知の通り、外部メモリを高速にアクセスするための小容量高速メモリであるバッファから成る機構である。一般にプログラムは、メモリに配置された関数をアドレス順に処理していく。しかし、外部メモリのメモリアクセス速度はCPUの処理速度に比べて非常に遅いため、結果として実行速度が遅くなるという問題がある。

【0004】上記の問題点を解消するため、関数実行の際に、外部メモリに格納してあるプログラムを、高速アクセス可能なバッファであるキャッシュにコピーして実行することにより、高速なプログラム実行を実現することが可能となる。この理由は、一般に、プログラムを実行すると、一度アクセスされたメモリは、近いうちに再度アクセスされる可能性が高いという性質があるからである。

【0005】ただし、一般的にキャッシュ用の高速メモリは外部メモリに比較して高価であり、その構成にはコストがかかるため、外部メモリに比べて非常に容量（サイズ）は小さい。このため、キャッシュを用いる処理システムは、外部メモリをキャッシュのサイズで区切った領域に分割し、分割した領域毎に外部メモリをキャッシュに割り当てる。あるアドレスに対する最初のアクセスでそのアドレスのプログラムを上記キャッシュ割り当て領域にコピーし、次に外部メモリの同一アドレスをアクセスした場合はキャッシュを直接アクセスする。このことで、高速なプログラム実行を実現する。

【0006】このとき、外部メモリからキャッシュメモリへのコピーは特定のサイズの単位で行われ、このサイズでキャッシュを分割した領域をキャッシュラインと呼ぶ。従って、同一のキャッシュラインに割り当てられた外部メモリのアドレスに配置された関数同士は、関数が切り替わる度に、キャッシュにプログラムををコピーし直す必要が生じてくる。これをキャッシュコンフリクトという。このキャッシュコンフリクトが頻繁に起きる

と、結果としてプログラムの実行速度が遅くなってしまうという問題がある。よって、昨今では、この問題を解消すべく同時に動く可能性の高い関数同士は、同一のキャッシュラインには載らないように配置する方法が研究されている。

【0007】なお、キャッシュには命令キャッシュとデータキャッシュがあるが、本発明は、命令キャッシュに着目するものである。

【0008】外部メモリのキャッシュへの割当て方式には、もっとも単純で安価なダイレクトマップ方式や、セットアソシアティブ／フルアソシアティブといった方式があるが、基本的な問題は全て同じであるため、以降、ダイレクトマップ方式を例にとり説明する。

【0009】また、従来の言語処理系プログラムでは、ある処理単位（関数）毎にメモリに適当に配置していた。このため、キャッシュ搭載のシステムにおいては、それが必ずしも最適に利用されているとは限らなかった。

【0010】最新の従来技術では、命令キャッシュに関し、キャッシュコンフリクトの回数を確率的に減らすために、関数の呼び出し回数情報に基づいて関数のメモリ配置を最適化することにより、キャッシュを有効に利用する研究がなされており、いくつかのアルゴリズムが論文発表されている。

【0011】例えば、エー・エッチ・ハッシェミ、デー・アール・カエリ、ビー・カルダ、「キャッシュラインカラーリングを用いた効率的マッピング手順」（A. H. Hashemi, D. R. Kaeli, B. Calder" EfficientProcedure Mapping Using Cache Line Colorling"） ACM SIGPLAN、1997年6月、（文献1）においては、呼び出し回数の多い関数から順にキャッシュコンフリクトを避けるように関数のメモリ配置を最適にしていく手法が公開されている。

【0012】また、特開平11-232117号公報（文献2）記載の従来の第1の命令キャッシュへの関数割付最適化方法においては、キャッシュメモリを効率よく使用するための関数配置方法の実施例として、上記文献1の方法が採用され、詳しく説明されている。

【0013】本従来技術のアルゴリズムでは、関数の呼出グラフを作成し、呼び出し回数をその辺（関数の組み合わせ）に対する重みとして優先順位をつけてメモリ空間に配置する。このことにより、まず関数を最初に配置した時のキャッシュコンフリクトを避けることができる。さらに、各関数が配置された「使用色」（すなわち、使用されているキャッシュライン）と、その関数が現在利用できない「利用不可色」の集合を記録しておき、後者、すなわち、利用不可色を使わないように関数を配置し、既に配置した関数についても、その関数の利用できない利用不可色を使わないという条件の下で、別の場所に移動する。これにより、直接の「親」あるいは「子」との間で発生するキャッシュコンフリクトを除去するものである。

【0014】次に、後述する本発明の実施の形態で適用するアプリケーションプログラムに対する従来例での動作を確認する。このことにより、従来例における問題点をより詳細に説明する。

【0015】従来の第1の命令キャッシュへの関数割付最適化装置をブロックで示す図10を参照すると、この従来の第1の命令キャッシュへの関数割付最適化装置は、アプリケーションプログラム110から関数呼出情報を読み込み、関数呼び出し時に呼出元と呼出先の各関数情報とその呼出回数を関数呼出組合せ情報111に出力する関数呼出情報出力部1と、関数呼出組合せ情報111に基づき関数の配置を最適化してアドレス空間に配置し関数メモリ配置結果104を出力する関数メモリ配置最適化部103とを備える。

【0016】図10、関数メモリ配置最適化部103の処理フローをフローチャートで示す図11、及び上記アプリケーションプログラムの一例を示す図3を参照して、従来の第1の命令キャッシュへの関数割付最適化装置の動作である従来の第1の命令キャッシュへの関数割付最適化方法について説明すると、まず、図3に示すアプリケーションプログラム110を適用した場合、関数呼び出し情報出力部1は、プロファイルにより関数呼び出し時に呼出元と呼出先の各関数情報とその呼出回数を関数呼出組合せ情報111に出力する。なお、図10及び図11において、実線の矢印は処理の流れを示し、点線の矢印はデータの流れを示す。

【0017】関数呼出組合せ情報111の一例を示す図2を参照すると、この関数呼出組合せ情報111は、関数の呼出元、呼出先、呼出回数の各欄から成る。

【0018】次に、関数メモリ配置最適化部103は、関数呼出組合せ情報111を呼出回数の多い順にソートし、この順番にアドレス空間に配置すると同時に配置した関数が利用できないキャッシュライン対応の「利用不可色」の集合を認識し、これを避けて後続の関数を配置する。

【0019】すなわち、図9において、ステップP1では関数呼出組合せ情報111から図12に示す関数呼出グラフ120を作成し、ステップP2では作成した関数関数呼出グラフ120を呼び出し回数の多いものと少ないものとに分割する。ここでは、func-func A, func-funcB, func-funcCが前者の「多いもの」、main-func, funcA-funcD, funcB-funcD、が後者の「少ないもの」となる。

【0020】ここで作成した関数呼出グラフ120においては、関数の組合せが辺となり、その両端のノードが

組合せにおける２つの関数となる。

【００２１】なお、図３における各関数の占めるキャッシュライン数すなわち「色」の数は、ｆｕｎｃが２個、ｆｕｎｃＡ，ｆｕｎｃＢ，ｆｕｎｋＣ，ｍａｉｎが各１個であるものとする。

【００２２】次に、ステップＰ３において、呼出回数の多いもののグループを呼出回数の多い順にソートし、その順番でステップＰ４以降の処理を行う。ステップＰ４では呼出回数の多い辺が残っているか確認し、残っているのでステップＰ５に進み、ｆｕｎｃ－ｆｕｎｃＡの辺に対して両側のノードが未配置であるかを確認する。この確認において未配置であるのでステップＰ９に進み、ｆｕｎｃとｆｕｎｃＡをメモリ空間上の任意の場所に隣接して配置し、ステップＰ１５においてｆｕｎｃとｆｕｎｃＡの利用できない「色」を利用不可能集合として認識した後、再びステップＰ４に戻る。隣接して配置されたｆｕｎｃ－ｆｕｎｃＡの辺は、複合ノードとして今後ひとつのノードとして扱われる。この時点で、既に配置済みの関数とキャッシュラインの関係および各関数の利用不可能集合の状態は、図１３（Ａ）に示すようになっている。

【００２３】続いて、呼出回数の多い辺がまだ残っているのでステップＰ５に進み、ｆｕｎｃ－ｆｕｎｃＣの辺に対して両側のノードが未配置であるかを確認する。この確認において、ｆｕｎｃは配置済みであるのでステップＰ６に進み、２個の異なる複合ノードに属するノードを結ぶ辺かどうかを確認する。この確認において、ｆｕｎｃＣは複合ノードに属していないので、ステップＰ７に進み、一方のノードが複合ノードに属し他方のノードが未配置かどうか確認すると、条件に当てはまるのでステップＰ１１に進む。

【００２４】ステップＰ１１では未配置のｆｕｎｃＣをｆｕｎｃに近い場所に配置し、ステップＰ１３において関数配置の際に利用不可能集合の影響で隙間が空いてないかを確認すると空いていないので、ステップＰ１５においてｆｕｎｃとｆｕｎｃＣを利用できない「利用不可色」を利用不可能集合として認識した後に、再びステップＰ４に戻る。

【００２５】この時点で、既に配置済みの関数とキャッシュラインの関係及び各関数の利用不可能集合の状態は、図１３（Ｂ）に示すようになっている。

【００２６】ステップＰ４において、まだ未配置の辺があるので、ステップＰ５に進み、ｆｕｎｃ－ｆｕｎｃＢの辺に対し、両側のノードが未配置であるかを確認する。この確認において、ｆｕｎｃは配置済みであるので、ステップＰ６に進み、２個の異なる複合ノードに属するノードを結ぶ辺かどうかを確認する。この確認において、ｆｕｎｃＢは複合ノードに属していないので、ステップＰ７に進み、一方のノードが複合ノードに属し他方のノードが未配置かどうか確認する。この確認において、条件に当てはまるので、ステップＰ１１に進む。

【００２７】ステップＰ１１では、未配置のｆｕｎｃＢと対を成すｆｕｎｃの中心から複合ノードの両端までの距離が同じであるため、任意に左側に配置し、ステップＰ１３において関数配置の際、利用不可能集合の影響で隙間が空いてないか確認する。すると、空いていないので、ステップＰ１５に進み、ｆｕｎｃとｆｕｎｃＢの利用できない「利用不可色」を利用不可能集合として認識したのち、再びステップＰ４に戻る。

【００２８】ステップＰ４において、未配置の辺が無くなったことを確認すると、ステップＰ１６に進み、未配置ノードｍａｉｎ、ｆｕｎｃＤを任意のキャッシュラインに配置する。

【００２９】図１３（Ｃ）は最終的な関数配置とキャッシュラインの関係、及び各関数の利用不可能集合の状態、すなわち、関数メモリ配置結果１０４であるが、ｆｕｎｃＡ、ｆｕｎｃＢが同一キャッシュライン「青」を共有しており、それぞれの利用不可能集合には「青」が含まれていない。よって、呼び出し元関数と呼び出し先関数の間のキャッシュコンフリクトは削減できる。

【００３０】上記の従来の第１の技術では、手続き、関数、あるいはサブルーチン同士が、互いを呼び出す際のキャッシュメモリ上での衝突およびキャッシュミスを防止することが目的である。この目的において、手続き、関数、あるいはサブルーチンが実際に呼び出される回数を示す情報と、手続き、関数、あるいはサブルーチン同士が、互いを呼び出す関係を示す情報とを利用している。これにより、手続き、関数、あるいはサブルーチン同士が、互いを呼び出す際のキャッシュメモリ上での衝突を防止できる。

【００３１】このように、従来の第１の技術においては呼出元関数と呼出先関数の間のキャッシュコンフリクトは削減できるが、（１）ある関数の中で複数の関数が連続して呼ばれている場合、あるいは（２）ループの中で呼ばれている場合等には、これら複数の関数間のキャッシュコンフリクトを削減できず、極めて多くのキャッシュコンフリクトが生じてしまうという第１の問題がある。

【００３２】つまり、ｆｕｎｃＡとｆｕｎｃＢは直接の呼出関係がないため、関数呼出組み合わせ情報を元にした関数配置を行う従来技術では、これらの関数が同一キャッシュラインに乗ってしまう場合があり得る。しかし、図３の上記アプリケーションプログラム例より明らかな通り、ｆｕｎｃＡとｆｕｎｃＢはループ中で連続して呼ばれ、さらにｆｕｎｃＢでは、ｆｕｎｃＤを経由してｆｕｎｃＡを呼び出すというプログラム記述となっており、ｆｕｎｃＡとｆｕｎｃＢが頻繁に遷移を繰り返すため、このループ処理において、極めて多くのキャッシュコンフリクトが生じてしまう。

【００３３】また、以上の第１の問題を解決するため、

特願２０００－０２７２１８号明細書（文献３）記載の従来の第２の命令キャッシュへの関数割付最適化方法は、プロファイルにより直接の関数呼出組み合わせ情報を出力する代わりに、関数実行の時系列情報を出力し、この時系列情報から、連続した関数呼出しなど直接の関数呼出し以外にキャッシュコンフリクトを発生する可能性のある関数の組み合わせ実行パターンを検出し、検出した関数間キャッシュコンフリクト組み合わせ情報に対して、従来技術の関数配置最適化を適用する。これにより、従来削減できなかった、（１）ある関数の中で複数の関数が連続して呼ばれている場合、あるいは（２）ループの中で呼ばれている場合など、これら複数の関数間のキャッシュコンフリクトを削減し、アプリケーションプログラムの実行スピードを向上する手段を提供するものである。

【００３４】しかしながら、この従来の第２の技術は、プログラムの遷移が単純な場合は、単純な処理で実現可能であり、極めて有効な手段であるが、この例で示したように、（３）ループ中で呼ばれる関数がまた別の関数を呼んでいるような場合などでは、パターンマッチングができなくなり、最適化ができないという第２の問題がある。

【００３５】

【発明が解決しようとする課題】上述した従来の第１の命令キャッシュへの関数割付最適化装置、関数割付最適化方法及び関数割付最適化手順を記録した記録媒体は、ある関数の中で複数の関数が連続して呼ばれている場合、あるいはループの中で呼ばれている場合等には、これら複数の関数間のキャッシュコンフリクトを削減できるとは限らず、最悪の場合には極めて多くのキャッシュコンフリクトが生じてしまうというという欠点があった。

【００３６】また、上記欠点の解決を図った従来の第２の命令キャッシュへの関数割付最適化装置、関数割付最適化方法及び関数割付最適化手順を記録した記録媒体は、プログラムの遷移が単純な場合は、単純な処理で実現可能であり、極めて有効な手段であるが、ループ中で呼ばれる関数がまた別の関数を呼んでいるような場合などでは、パターンマッチングができなくなり、最適化が不可能となるという欠点があった。

【００３７】本発明の目的は、上記第１及び第２の従来技術の欠点を除去し、複数の関数間のキャッシュコンフリクトを削減し、アプリケーションプログラムの実行スピードの向上を図った命令キャッシュへの関数割付最適化装置、関数割付最適化方法及び関数割付最適化手順を記録した記録媒体を提供することにある。

【００３８】

【課題を解決するための手段】請求項１の発明の命令キャッシュへの関数割付最適化装置は、命令キャッシュを搭載したマイクロプロセッサシステム用の所定のアプリケーションプログラムを入力し前記命令キャッシュに関しキャッシュコンフリクトの回数を確率的に低減するように関数の呼出回数情報に基づいて関数のメモリ配置を最適化する命令キャッシュへの関数割付最適化装置において、前記アプリケーションプログラムを入力しプロファイルによる関数呼出時に呼出元及び呼出先の各関数とその呼出回数とを関数呼出組合せとして関数呼出組合せ情報に出力する関数呼出情報出力部と、前記アプリケーションプログラムを入力し前記プロファイルによる関数呼出に応じた関数の遷移に対して該関数のＩＤ及び該関数の基本ブロックの順番の組合せを関数遷移毎に並べた関数基本ブロック遷移情報に出力する関数基本ブロック遷移情報出力部と、前記関数呼出組合せ情報を参照し前記関数呼出組合せ相互間で前記関数遷移の回数である遷移回数を入替えた関数呼出回数入替データから成る呼出回数入替情報を生成し、次に生成した前記呼出回数入替情報を参照して関数をメモリ空間上のアドレスに仮配置した後、前記関数基本ブロック遷移情報を参照してキャッシュコンフリクトの回数を検出し、前記呼出回数入替データの中で、前記キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定し対応する関数メモリ配置結果を出力する関数メモリ配置最適化部とを備えて構成されている。

【００３９】また、前記関数呼出組合せ情報及び前記呼出回数入替情報の各々が、関数の呼出元の関数名を記述した呼出元欄と、前記関数の呼出先の関数名を記述した呼出先欄と、前記関数の呼出回数を設定する呼出回数欄とをそれぞれ有しても良い。

【００４０】請求項３の命令キャッシュへの関数割付最適化方法は、命令キャッシュを搭載したマイクロプロセッサシステム用のアプリケーションプログラムを入力し前記命令キャッシュに関しキャッシュコンフリクトの回数を確率的に低減するように関数の呼出回数情報に基づいて関数のメモリ配置を最適化する命令キャッシュへの関数割付最適化方法において、前記アプリケーションプログラムを入力し前記プロファイルによる関数呼出時に呼出元及び呼出先の各関数とその呼出回数とを関数呼出組合せとして関数呼出組合せ情報を生成し、前記アプリケーションプログラムを入力し前記プロファイルにより得られた前記関数の基本ブロック単位の実行に関する関数基本ブロック遷移情報を生成した後、前記関数呼出組合せ情報を参照し前記関数呼出組合せ相互間で前記関数遷移の回数である遷移回数を入替えた関数呼出回数入替データから成る呼出回数入替情報を生成し、前記関数基本ブロック遷移情報を参照して各関数のキャッシュコンフリクトの回数を検出し、前記呼出回数入替データの中で前記キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定する関数メモリ配置最適化工程を有することを特徴とするものである。

【００４１】また、前記関数メモリ配置最適化工程が、

前記関数呼出組合せ情報を参照して前記呼出回数入替情報を生成する呼出回数入替ステップと、生成した前記呼出回数入替情報を参照して関数をメモリ空間上のアドレスに仮配置する関数メモリ仮配置ステップと、前記関数基本ブロック遷移情報を参照して仮配置した各関数のキャッシュコンフリクト回数を検出するキャッシュコンフリクト回数算出処理ステップと、前記関数呼出回数入替データの中で前記キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定する関数メモリ配置ステップとを有することを特徴としても良い。

【００４２】また、前記関数メモリ配置最適化工程における前記呼出回数入替ステップが、前記関数呼出組合せ情報を参照して呼出回数の入替対象とする前記関数呼出組合せの数である引数を読み込み第１の変数を設定する第１のステップと、前記第１の変数が０か否かを判定する第２のステップと、前記第２のステップで前記第１の変数が０の場合現在の内容の呼出回数入替情報を出力する第３のステップと、前記第２のステップで前記第１の変数が０以外の場合呼出回数入替処理を行い前記第１の変数－１に対応する引数を設定して再帰呼出を行う第４のステップと、第２の変数に０を設定する第５のステップと、前記第２の変数が前記第１の変数－１より小さいか否かの判定を行い否の場合は処理を終了する第６のステップと、前記第６のステップで諾の場合前記第２の変数と前記第１の変数－１である第１のインッデクスの各々の前記呼出回数を交換する第７のステップと、前記引数を１デクリメントして前記呼出回数入替処理を行い前記第１の変数－１に対応する引数を設定して再帰呼出を行う第８のステップと、前記第２の変数である第２のインデックスと前記第１の変数－１の各々の前記呼出回数を交換する第９のステップと、前記第２の変数を１インクリメントし前記第６のステップ以降を反復する第１０のステップとを有することを特徴としても良い。

【００４３】さらに、前記関数メモリ配置最適化工程における前記キャッシュコンフリクト回数算出処理ステップが、キャッシュコンフリクト回数をカウントする変数を０に初期化する第１のステップと、前記関数基本ブロック遷移情報を順次読み込み、関数ＩＤと基本ブロックの順番情報であるＩＤ順番情報を求める第２のステップと、前記関数基本ブロック遷移情報は終了したかを判定し諾の場合は処理を終了する第３のステップと、前記第３のステップで否の場合は前記ＩＤ順番情報のキャッシュ上の配置を求める第４のステップと、先頭の前記関数基本ブロック遷移情報かを判定し諾の場合は前記第２のステップに戻る第５のステップと、前記第５のステップで否の場合以前のブロックとアドレス上の重なりがあるかを判定し否の場合は前記第２のステップに戻る第６のステップと、前記第６のステップで諾の場合は前記キャッシュコンフリクト回数をカウントする変数を１インクリメントし前記第２のステップに戻る第７のステップとを有することを特徴としても良い。

【００４４】請求項７の命令キャッシュへの関数割付最適化手順を記録した記録媒体は、命令キャッシュを搭載したマイクロプロセッサシステム用のアプリケーションプログラムを入力し前記命令キャッシュに関しキャッシュコンフリクトの回数を確率的に低減するように関数の呼出回数情報に基づいて関数のメモリ配置を最適化する命令キャッシュへの関数割付最適化手順を記録した記録媒体において、前記アプリケーションプログラムを入力し前記プロファイルによる関数呼出時に呼出元及び呼出先の各関数とその呼出回数とを関数呼出組合せとして関数呼出組合せ情報を生成する手順と、前記アプリケーションプログラムを入力し前記プロファイルにより得られた前記関数の基本ブロック単位の実行に関する関数基本ブロック遷移情報を生成する手順と、前記関数呼出組合せ情報を参照し前記関数呼出組合せ相互間で前記関数遷移の回数である遷移回数を入替えた関数呼出回数入替データから成る呼出回数入替情報を生成し、前記関数基本ブロック遷移情報を参照して各関数のキャッシュコンフリクトの回数を検出し、前記呼出回数入替データの中で前記キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定する関数メモリ配置最適化手順とを実行させるプログラムを記録したことを特徴とするものである。

【００４５】

【発明の実施の形態】次に、本発明の実施の形態について図面を参照して説明する。

【００４６】本発明は、命令キャッシュを搭載したマイクロプロセッサシステム用のアプリケーションプログラムを入力し、上記命令キャッシュに関しキャッシュコンフリクトの回数を確率的に低減するように関数の呼出回数情報に基づいて関数のメモリ配置を最適化する命令キャッシュへの関数割付最適化方法において、プロファイルによる関数呼出時に呼出元及び呼出先の各関数とその呼出回数とを関数呼出組合せとして関数呼出組合せ情報を生成し、プロファイルにより得られた関数の基本ブロック単位の実行に関する関数基本ブロック遷移情報を生成し、関数呼出組合せ情報を参照し関数呼出組合せ相互間で関数遷移の回数である遷移回数を入替えた関数呼出回数入替データから成る呼出回数入替情報を生成し、関数基本ブロック遷移情報を参照して命令キャッシュのキャッシュコンフリクト回数を検出して、呼出回数入替データの中で最もキャッシュコンフリクト回数の少なくなるように関数をメモリ空間上のアドレスに配置することにより、プログラムの実行スピードを向上させるものである。

【００４７】本発明の実施の形態を図１０と共通の構成要素には共通の参照文字／数字を付して同様にブロックで示す図１を参照すると、この図に示す本実施の形態の命令キャッシュへの関数割付最適化装置は、従来と共通

のアプリケーションプログラム１１０を入力し、プロファイルによる関数呼出時に呼出元及び呼出先の各関数とその呼出回数とを関数呼出組合せとして関数呼出組合せ情報１１１に出力する関数呼出情報出力部１に加えて、アプリケーションプログラム１１０を入力し、プロファイルによる関数呼出に応じた関数の遷移に対して、関数のＩＤ（識別）及びその関数の基本ブロックの順番の組合せを関数遷移毎に並べた関数基本ブロック遷移情報１１２に出力する関数基本ブロック遷移情報出力部２と、関数呼出組合せ情報１１１を参照して関数呼出組合せ相互間で上記関数遷移の回数である遷移回数を入替えて関数呼出回数入替データから成る呼出回数入替情報１１３を生成し、次に生成した呼出回数入替情報１１３を参照して関数をメモリ空間上のアドレスに仮配置した後、関数基本ブロック遷移情報１１２を参照してキャッシュコンフリクトの回数を検出し、関数呼出回数を入替えたもの、すなわち、関数呼出回数入替データの中で、キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定し対応する関数メモリ配置結果４を出力する関数メモリ配置最適化部３とを備える。

【００４８】なお、図１において、実線の矢印は処理の流れを示し、点線の矢印はデータの流れを示す。

【００４９】関数呼出組合せ情報１１１の一例を示す図２を参照すると、この関数呼出組合せ情報１１１は、関数の呼出元の関数名を記述した呼出元欄、関数の呼出先の関数名を記述した呼出先欄、及びその呼出回数を設定する呼出回数欄の各欄から成る。

【００５０】また、呼出回数入替情報１１３も、その構成は関数呼出組合せ情報１１１と全く同じであり、関数の呼出元の関数名を記述した呼出元欄、関数の呼出先の関数名を記述した呼出先欄、及びその呼出回数を設定する呼出回数欄の各欄から成る。

【００５１】次に、図１、図２、上記アプリケーションプログラムの一例を示す図３、及び関数メモリ配置最適化部の処理フローをフローチャートで示す図５を参照して本実施の形態の動作について説明すると、まず、関数呼出情報出力部１は、アプリケーションプログラム１１０を入力し、従来と同様に、プロファイルにおける関数呼び出し時に呼出元と呼出先の各関数情報とその呼出回数を関数呼出組合せ情報１１１に出力する。ここで、プロファイルとは、関数の基本ブロックの先頭及び呼出関数からの復帰時に、関数ＩＤと基本ブロックの順番を出力するコードをアプリケーションプログラム１１０に挿入して実行することである。

【００５２】次に、関数基本ブロック遷移情報出力部２は、アプリケーションプログラム１１０を入力し、プロファイル、すなわち、関数の基本ブロックの先頭、及び呼出関数から復帰時に、関数ＩＤと基本ブロックの順番を出力するコードをアプリケーションプログラム１１０に挿入して実行することにより、これらの情報を関数基本ブロック遷移情報１１２に出力する。

【００５３】図３を再度参照すると、この図に示すアプリケーションプログラム１１０はＣ言語によるアプリケーションプログラムの例であり、関数ｍａｉｎの関数ＩＤを０、関数ｆｕｎｃの関数ＩＤを１、関数ｆｕｎｃＡの関数ＩＤを２、関数ｆｕｎｃＢの関数ＩＤを３、関数ｆｕｎｃＣの関数ＩＤを４、関数ｆｕｎｃＤの関数ＩＤを５とそれぞれ想定する。

【００５４】さらに関数ｆｕｎｃは、関数の基本処理単位である基本ブロックがループ文２つから構成され、また、関数ｍａｉｎ，ｆｕｎｃＡ，ｆｕｎｃＢ，ｆｕｎｃＣ，ｆｕｎｃＤの各々の基本ブロックが各１つの構成とする。

【００５５】まず、先頭の関数ｍａｉｎの呼出時点でｍａｉｎのＩＤである０と基本ブロックの１番目の組合せである０－１（以下、組合せ０－１等）とを出力する。次に、関数ｍａｉｎから関数ｆｕｎｃに処理が移り、ｆｕｎｃのＩＤである１と基本ブロック１番目の組合せ１－１とを出力する。次に、関数ｆｕｎｃからｆｕｎｃＡに処理が移り、ｆｕｎｃＡの関数ＩＤである２と基本ブロック１番目の組合せ２－１とを出力する。次に、関数ｆｕｎｃＡからｆｕｎｃに処理が復帰し、ｆｕｎｃのＩＤである１と基本ブロックの１番目の組合せ１－１とを出力する。以降、同様にしてプログラムの終了まで関数ＩＤと基本ブロックの順番の組合せを出力する。

【００５６】本例のアプリケーションプログラム１１０は、関数ｍａｉｎから関数ｆｕｎｃを呼出し、関数ｆｕｎｃでは関数ｆｕｎｃＡと関数ｆｕｎｃＢを連続して呼出す処理を２０回繰返す（反復）処理と、続いて関数ｆｕｎｃＡと関数ｆｕｎｃＣを連続して呼出す処理を３０回繰返す処理を行い、関数ｍａｉｎに戻って終了するプログラムである。ここで、関数ｆｕｎｃＢは関数ｆｕｎｃＤを呼出し、さらに、関数ｆｕｎｃＤは関数ｆｕｎｃＡを呼出す構成である。この処理により、プログラムの終了までプロファイルを実行すると、図４に示すような関数ＩＤと基本ブロックの順番の情報の配列である関数基本ブロック遷移情報１１２が作成される。

【００５７】次に、関数メモリ配置最適化部３は、関数呼出組合せ情報１１１を参照して呼出回数を入替えて関数呼出回数入替データから成る呼出回数入替情報１１３を生成する。次に、生成した呼出回数入替情報１１３を参照して、関数をメモリ空間上のアドレスに仮配置する。その後、関数基本ブロック遷移情報１１２を参照して仮配置した各関数のキャッシュコンフリクトの回数を検出し、呼出回数を入替たもの、すなわち、関数呼出回数入替データの中で、キャッシュコンフリクトの最も回数の少ないものに関数のメモリ配置を決定する。

【００５８】すなわち、図５を併せて参照して関数メモリ配置最適化部３の動作の詳細を説明すると、まず関数呼出組合せ情報１１１を参照して呼出回数入替処理ステ

ップS1を行う。本処理では引数として呼出回数の多い組合せの数を与える。この多い少ないの基準は、従来技術である関数メモリ配置最適化部おける関数呼出グラフ分割と同様である。本例では関数呼出組合せ情報111において、func-funcA、func-funcB、func-funcCの各組合せを関数呼出回数が多いと判断し、その他の組合せを少ないと判断する。よって呼出回数の多い組合せの数は3種類となり、これを引数として渡すこととなる。以後、呼出回数の入替対象は、この3組となる。

【0059】ここで、呼出回数入替処理ステップS1の詳細をフローチャートで示す図6及び呼出回数入替情報113の一例を示す図7を併せて参照して呼出回数入替処理ステップS1の詳細動作について説明すると、まず、ステップS101で引数3を読み込む。次に、ステップS102で変数nに引数3を設定する。次に、ステップS103の条件判定「nが0か」を行い、変数nが0でないので、ステップS105に分岐し、変数n-1とする呼出回数入替処理を行い、n-1に対応する引数2として再帰呼出を行う。

【0060】次に、再度ステップS101で引数2を読み込み、ステップS102で変数nに2を設定する。次に、ステップS103の条件判定を行い、変数nが0でないので、ステップS105に分岐し、変数n-1とする呼出回数入替処理を行い、引数1として再び再帰呼出を行う。

【0061】次に、ステップS101で引数1を読み込み、ステップS102で変数nに1を設定する。ステップS103の条件判定を行い、変数nが0でないので、ステップS105に分岐し、変数n-1とする呼出回数入替処理を行い、引数0としてさらに再帰呼出を行う。

【0062】次に、ステップS101で引数0を読み込み、ステップS102で変数nに引数0を設定する。ステップS103の条件判定を行い、変数nが0であるので、ステップS104に進み、現在の内容である呼出回数入替情報113を出力する。ここまでは呼出回数は全く入替えていないので、呼出回数入替情報113として図7（A）に示す関数呼出組合せ情報111と全く同じものを出力し、引数0における処理は終了する。

【0063】次に、引数1における処理に戻り、ステップS106で変数iに初期値0を設定し、ステップS107の条件判定「i＜(n-1)」を行う。n-1は0であり従って変数iは(n-1)より小さくはないので、引数1における処理は終了する。

【0064】次に、引数2における処理に戻り、ステップS106で変数iに初期値0を設定し、ステップS107の条件判定を行う。n-1は1であり従って変数iは(n-1)より小さいので、ステップS108に進む。ステップS108で、インデックスn-1及びiの要素、すなわち、呼出回数を交換する。この例では、インデックス0(i)対応のfunc-funcAの呼出回数50と、インデックス1(n-1)対応のfunc-funcBの呼出回数20とを交換する。その後、ステップS109で引数を1として再び再帰呼出を行う。

【0065】次に、ステップS101で引数1を読み込み、ステップS102で変数nに1を設定する。ステップS103の条件判定を行い、変数nが0でないので、ステップS105に分岐し、呼出回数入替処理ステップS109を引数0としてさらに再帰呼出を行う。

【0066】次にステップS101で、引数0を読み込み、ステップS102で変数nに引数0を設定する。ステップS103の条件判定を行い、変数nが0であるので、ステップS104に進み、図7（B）に示す呼出回数入替情報113を出力し、引数0における処理は終了する。

【0067】次に、引数1における処理に戻り、ステップS106で変数iに初期値0を設定し、ステップS107の条件判定「i＜(n-1)」を行う。n-1は0であり従って変数iは(n-1)より小さくはないので、引数1における処理を終了する。

【0068】次に、引数2における処理に戻り、ステップS110でインデックス0対応のfunc-funcAの呼出回数20と、インデックス1対応のfunc-funcBの呼出回数50とを交換して元に戻す。その後、ステップS110で変数iを0から1とし、ステップS107の条件判定を行う。n-1は0であり従って変数iは(n-1)より小さくはないので、引数2における処理を終了する。

【0069】以降、同様の処理を繰り返し、図7（C）、（D）、（E）、（F）をそれぞれ出力して、呼出回数入替処理ステップS1を終了する。

【0070】次に、図5に戻り、ステップS2で、呼出回数入替情報113を参照し、この情報が（A）から（F）までの6個出力されていることを認識し、（A）から順に以後の処理を行っていく。

【0071】以降、ステップS4からステップS19まで、従来のステップP1～P16と同一処理を行う。

【0072】このステップS4～S19のアルゴリズムは、従来の技術で説明したように、関数の呼出グラフを作成し、呼び出し回数をその辺（関数の組み合わせ）に対する重みとして優先順位をつけてメモリ空間に配置することにより、関数を最初に配置した時のキャッシュコンフリクトを避けることができる。各関数が配置された「使用色」（すなわち、使用されているキャッシュライン）と、その関数が現在利用できない「利用不可色」の集合を記録しておき、後者、すなわち、利用不可色を使わないように関数を配置し、既に配置した関数についても、その関数の利用できない利用不可色を使わないという条件の下で、別の場所に移動する。これにより、直接の「親」あるいは「子」との間で発生するキャッシュコ

ンフリクトを除去するものである。

【００７３】まず、ステップＳ４で、関数呼出組合せ情報１１１から図１２に示す関数呼出グラフ１２０を作成し、ステップＳ５では作成した関数関数呼出グラフ１２０を呼出回数の多いものと少ないものとに分割する。ここでは、ｆｕｎｃ－ｆｕｎｃＡ，ｆｕｎｃ－ｆｕｎｃＢ，ｆｕｎｃ－ｆｕｎｃＣの各組合せが呼出回数の「多いもの」、ｍａｉｎ－ｆｕｎｃ，ｆｕｎｃＡ－ｆｕｎｃＤ，ｆｕｎｃＢ－ｆｕｎｃＤ、が呼出回数の「少ないもの」となる。

【００７４】ここで作成した関数呼出グラフ１２０においては、関数の組合せが辺となり、その両端のノードが組合せにおける２つの関数となる。

【００７５】なお、図3における各関数の占めるキャッシュライン数、すなわち「色」の数は、ｆｕｎｃが２個、ｆｕｎｃＡ，ｆｕｎｃＢ，ｆｕｎｋＣ，ｍａｉｎが各１個であるものとする。

【００７６】次に、ステップＳ６において、呼出回数の多いもののグループを呼出回数の多い順にソートし、その順番でステップＳ７以降の処理を行う。ステップＳ７では呼出回数の多い辺が残っているか確認し、残っているのでステップＳ８に進み、ｆｕｎｃ－ｆｕｎｃＡの辺に対して両側のノードが未配置であるかを確認する。この確認において未配置であるのでステップＳ９に進み、ｆｕｎｃとｆｕｎｃＡをメモリ空間上の任意の場所に隣接して配置し、ステップＳ１８においてｆｕｎｃとｆｕｎｃＡの利用できない「色」を利用不可能集合として認識した後、再びステップＳ７に戻る。隣接して配置されたｆｕｎｃ－ｆｕｎｃＡの辺は、複合ノードとして今後ひとつのノードとして扱われる。この時点で、既に配置済みの関数とキャッシュラインの関係および各関数の利用不可能集合の状態は、図１３（Ａ）に示すようになっている。

【００７７】続いて、呼出回数の多い辺がまだ残っているのでステップＳ８に進み、ｆｕｎｃ－ｆｕｎｃＣの辺に対して両側のノードが未配置であるかを確認する。この確認において、ｆｕｎｃは配置済みであるのでステップＳ９に進み、２個の異なる複合ノードに属するノードを結ぶ辺かどうかを確認する。この確認において、ｆｕｎｃＣは複合ノードに属していないので、ステップＳ１０に進み、一方のノードが複合ノードに属し他方のノードが未配置かどうか確認すると、条件に当てはまるのでステップＳ１４に進む。

【００７８】ステップＳ１４では未配置のｆｕｎｃＣをｆｕｎｃに近い場所に配置し、ステップＳ１６において関数配置の際に利用不可能集合の影響で隙間が空いてないかを確認すると空いていないので、ステップＳ１８においてｆｕｎｃとｆｕｎｃＣを利用できない「利用不可色」を利用不可能集合として認識した後に、再びステップＳ７に戻る。

【００７９】この時点で、既に配置済みの関数とキャッシュラインの関係及び各関数の利用不可能集合の状態は、図１３（Ｂ）に示すようになっている。

【００８０】ステップＳ７において、まだ未配置の辺があるので、ステップＳ８に進み、ｆｕｎｃ－ｆｕｎｃＢの辺に対し、両側のノードが未配置であるかを確認する。この確認において、ｆｕｎｃは配置済みであるので、ステップＳ９に進み、２個の異なる複合ノードに属するノードを結ぶ辺かどうかを確認する。この確認において、ｆｕｎｃＢは複合ノードに属していないので、ステップＳ１０に進み、一方のノードが複合ノードに属し他方のノードが未配置かどうか確認する。この確認において、条件に当てはまるので、ステップＳ１４に進む。

【００８１】ステップＳ１４では、未配置のｆｕｎｃＢと対を成すｆｕｎｃの中心から複合ノードの両端までの距離が同じであるため、任意に左側に配置し、ステップＳ１６において関数配置の際、利用不可能集合の影響で隙間が空いてないか確認する。すると、空いていないので、ステップＳ１８に進み、ｆｕｎｃとｆｕｎｃＢの利用できない「利用不可色」を利用不可能集合として認識したのち、再びステップＳ７に戻る。

【００８２】ステップＳ７において、未配置の辺が無くなったことを確認すると、ステップＳ４６に進み、未配置ノードｍａｉｎ、ｆｕｎｃＤを任意のキャッシュラインに配置する。

【００８３】以上のステップＳ４～Ｓ１９の処理結果、図7（Ａ）に示す呼出回数入替情報１１３に対しては、従来の図１３（Ｃ）と同様の、本実施の形態の関数メモリ配置結果４を色で示す図９（Ａ）の配置となる。

【００８４】本実施の形態の関数配置とキャッシュラインの関係及び各関数の利用不可能集合の状態すなわち、関数メモリ配置結果４を示す図９を参照すると、図7に示す呼出回数入替情報１１３（Ａ）～（Ｆ）の各々に対応して図９（Ａ）～（Ｆ）に示すようになる。

【００８５】本実施の形態の例では、処理の対象とする６つの関数ｍａｉｎ、ｆｕｎｃ、ｆｕｎｃＡ、ｆｕｎｃＢ、ｆｕｎｃＣ、ｆｕｎｃＤのうち、呼出回数入替の対象とする４つの関数ｆｕｎｃ、ｆｕｎｃＡ、ｆｕｎｃＢ、ｆｕｎｃＣに利用不可能集合対応の色すなわち、利用不可能色を割り付ける。この例では、関数ｆｕｎｃに青及び黄を、関数ｆｕｎｃＡ、ｆｕｎｃＢ及びｆｕｎｃＣに赤及び緑をそれぞれ利用不可能色として割り付ける。

【００８６】次に、ステップＳ２０に進み、関数基本ブロック遷移情報１１２を参照してキャッシュコンフリクト回数算出処理を行う。

【００８７】以下の説明では、上述したように、関数ｍａｉｎ、ｆｕｎｃ、ｆｕｎｃＡ、ｆｕｎｃＢ、ｆｕｎｃＣ、及びｆｕｎｃＤの各々の関数ＩＤを０，１，２，３，４，５と設定してあるものとする。

【００８８】キャッシュコンフリクト回数算出処理ステップＳ２０の詳細をフローチャートで示す図８を参照してこのキャッシュコンフリクト回数算出処理の詳細動作について説明すると、まず、ステップＳ２０１で、コンフリクト回数をカウントする変数を０に初期化する。

【００８９】次に、関数基本ブロック遷移情報読み込みステップＳ２０２で、関数基本ブロック遷移情報１１２を順次読み込み、関数ＩＤと基本ブロックの順番情報（以下ＩＤ順番情報）０－１、すなわち、ｍａｉｎ－１番を得る。次に、ステップＳ２０３の条件判定「関数基本ブロック遷移情報は終了」で、関数基本ブロック遷移情報はまだ終了ではないと判定して、ステップＳ２０４に進む。ステップＳ２０４で、ＩＤ順番情報０－１のキャッシュ上の配置を求め、図９（Ａ）を参照して、関数ｍａｉｎ対応のキャッシュラインの色「黄」を得る。次に、ステップＳ２０５の条件判定「先頭の関数基本ブロック遷移情報か」で、ＩＤ順番情報０－１は先頭の関数基本ブロック遷移情報であるため、ステップＳ２０２に戻る。

【００９０】次に、再度ステップＳ２０２で、関数基本ブロック遷移情報１１２を順次読み込み、ＩＤ順番情報１－１、すなわち、ｆｕｎｃ－１番を得る。次に、ステップＳ２０３で、関数基本ブロック遷移情報はまだ終了ではないと判定して、ステップＳ２０４に進み、ＩＤ順番情報１－１のキャッシュ上の配置として、図９（Ａ）を参照して、関数ｆｕｎｃ対応のキャッシュラインの色「赤」を得る。次に、ステップＳ２０５の条件判定で、ＩＤ順番情報１－１は先頭の関数基本ブロック遷移情報ではないため、ステップＳ２０６に進む。次にステップＳ２０６の条件判定「以前のブロックとアドレス上の重なりがあるか」で、「赤」に配置されたものは以前にはなかったので、ステップＳ２０２に戻る。

【００９１】次に、再度ステップＳ２０２で、関数基本ブロック遷移情報１１２を順次読み込み、次のＩＤ順番情報２－１、すなわち、ｆｕｎｃＡ－１を得る。以下、上記処理と同様にステップＳ２０３で関数基本ブロック遷移情報はまだ終了ではないと判定して、ステップＳ２０４でＩＤ順番情報２－１のキャッシュ上の配置を、図９（Ａ）を参照して、関数ｆｕｎｃＡ対応のキャッシュライン「青」を得る。次に、ステップＳ２０５の条件判定で、このＩＤ順番情報２－１は先頭の関数基本ブロック遷移情報ではないため、ステップＳ２０６に進む。次にステップＳ２０６の条件判定で、「青」に配置されたものは以前にはなかったので、ステップＳ２０２に戻る。

【００９２】次に、ステップＳ２０２で、関数基本ブロック遷移情報１１２を順次読み込み、次のＩＤ順番情報１－１、すなわち、ｆｕｎｃ－１を得る。次に、ステップＳ２０３で関数基本ブロック遷移情報はまだ終了ではないと判定して、ステップＳ２０４でＩＤ順番情報１－１のキャッシュ上の配置として、図９（Ａ）を参照して、関数ｆｕｎｃ対応のキャッシュライン「赤」を得る。次にステップＳ２０５の条件判定で、ＩＤ順番情報１－１は先頭の関数基本ブロック遷移情報ではないため、ステップＳ２０６に進む。次にステップＳ２０６の条件判定で、「赤」に配置されたものは以前にはなかったので、ステップＳ２０２に戻る。

【００９３】次に、ステップＳ２０２で関数基本ブロック遷移情報１１２を順次読み込み、ＩＤ順番情報３－１、すなわち、ｆｕｎｃＢ－１を得る。次に、ステップＳ２０３で関数基本ブロック遷移情報はまだ終了ではないと判定して、ステップＳ２０４でＩＤ順番情報３－１のキャッシュ上の配置を、図９（Ａ）を参照して、関数ｆｕｎｃＢ対応のキャッシュライン「青」を得る。次に、ステップＳ２０５の条件判定で、ＩＤ順番情報３－１は先頭の関数基本ブロック遷移情報ではないため、ステップＳ２０６に進む。次に、ステップＳ２０６の条件判定で、「青」に配置された以前のブロックは今回と異なりＩＤ順番情報２－１であったので、ステップＳ２０７に進む。次に、ステップＳ２０７で、コンフリクト回数をカウントする変数（以下、コンフリクト回数）を１インクリメントし、ステップＳ２０２に戻る。

【００９４】次にステップＳ２０２で関数基本ブロック遷移情報１１２を順次読み込み、ＩＤ順番情報５－１、すなわち、ｆｕｎｃＤ－１を得る。次に、ステップＳ２０３で関数基本ブロック遷移情報はまだ終了ではないと判定して、ステップＳ２０４でＩＤ順番情報５－１のキャッシュ上の配置を、図９（Ａ）を参照して、関数ｆｕｎｃＤ対応のキャッシュライン「緑」と得る。次に、ステップＳ２０５の条件判定で、ＩＤ順番情報５－１は先頭の関数基本ブロック遷移情報ではないため、ステップＳ２０６に進む。次にステップＳ２０６の条件判定で、「緑」に配置されたものは以前にはなかったので、ステップＳ２０２に戻る。

【００９５】次に、ステップＳ２０２で関数基本ブロック遷移情報１１２を順次読み込み、ＩＤ順番情報２－１、すなわち、ｆｕｎｃＡ－１を得る。次に、ステップＳ２０３で関数基本ブロック遷移情報はまだ終了ではないと判定して、ステップＳ２０４で２－１のキャッシュ上の配置を、図９（Ａ）を参照して、関数ｆｕｎｃＡ対応のキャッシュライン「青」と得る。次に、ステップＳ２０５の条件判定で、ＩＤ順番情報２－１は先頭の関数基本ブロック遷移情報ではないため、ステップＳ２０６に進む。次に、ステップＳ２０６の条件判定で、「青」に配置された以前のブロックは今回と異なりＩＤ順番情報３－１であったので、ステップＳ２０７に進む。次に、ステップＳ２０７で、コンフリクト回数を１インクリメントし、ステップＳ２０２に戻る。

【００９６】次にステップＳ２０２で関数基本ブロック遷移情報１１２を順次読み込み、ＩＤ順番情報５－１、

すなわち、ｆｕｎｃＤ－１を得る。次に、ステップＳ２０３で関数基本ブロック遷移情報はまだ終了ではないと判定して、ステップＳ２０４でＩＤ順番情報５－１のキャッシュ上の配置を、図９（Ａ）を参照して、関数ｆｕｎｃＤ対応のキャッシュライン「緑」を得る。次に、ステップＳ２０５の条件判定で、ＩＤ順番情報５－１は先頭の関数基本ブロック遷移情報ではないため、ステップＳ２０６に進む。次に、ステップＳ２０６の条件判定で、「緑」に配置された以前のブロックは今回と同様ＩＤ順番情報５－１なので、ステップＳ２０２に戻る。

【００９７】次に、ステップＳ２０２で関数基本ブロック遷移情報１１２を順次読み込み、ＩＤ順番情報３－１、すなわち、ｆｕｎｃＢ－１を得る。次に、ステップＳ２０３で関数基本ブロック遷移情報はまだ終了ではないと判定して、ステップＳ２０４でＩＤ順番情報３－１のキャッシュ上の配置を、図９（Ａ）を参照して、関数ｆｕｎｃＢ対応のキャッシュライン「青」を得る。次に、ステップＳ２０５の条件判定で、ＩＤ順番情報３－１は先頭の関数基本ブロック遷移情報ではないため、ステップＳ２０６に進む。次に、ステップＳ２０６の条件判定で、「青」に配置された以前のブロックは今回と異なりＩＤ順番情報２－１だったので、ステップＳ２０７に進む。次に、ステップＳ２０７で、コンフリクト回数を１インクリメントし、ステップＳ２０２に戻る。

【００９８】以下、同様の処理を繰り返して関数基本ブロック遷移情報１１２の最後まで処理を行なうと、キャッシュライン「青」（以下、「青」等と省略）におけるコンフリクト回数は７９回、「黄」におけるコフリクト回数は２回、「緑」におけるコンフリクト回数は１回、「赤」におけるコンフリクト回数は０回となり、合計８２回のコンフリクトが発生することを算出する。

【００９９】以上により、キャッシュコンフリクト回数算出処理ステップＳ２０を終了する。

【０１００】次に、ステップＳ２に戻り、以後同様の処理を行い、図７に示す呼出回数入替情報１１３（図７（Ａ）～（Ｆ））の各々に対する関数配置とキャッシュラインの関係及び各関数の利用不可能集合の状態（以下キャッシュメモリ配置）はそれぞれ図９（Ａ）～（Ｆ）となる。

【０１０１】例えば、図９（Ｂ）のキャッシュメモリ配置の場合は、（Ａ）の場合と同一の８２回のコンフリクトが発生することを算出する。

【０１０２】また、図９（Ｃ）と（Ｆ）のキャッシュメモリ配置の場合は、「青」におけるコンフリクト回数は１回、「黄」におけるコフリクト回数は２回、「緑」におけるコンフリクト回数は１回、「赤」におけるコンフリクト回数は０回となり、合計４回のコンフリクトが発生することを算出する。

【０１０３】さらに、図９（Ｄ）と（Ｅ）のキャッシュメモリ配置の場合は、「青」におけるコンフリクト回数は７９回、「黄」におけるコフリクト回数は２回、「緑」におけるコンフリクト回数は１回、「赤」におけるコンフリクト回数は０回となり、合計８２回のコンフリクトが発生することを算出する。

【０１０４】よって、図９（Ｃ）と（Ｆ）のキャッシュメモリ配置の場合にキャッシュコンフリクトの発生回数が一番少ないことが分かる。

【０１０５】ステップＳ３において、関数のキャッシュメモリ配置をこのうちの一方である図９（Ｃ）の配置に最終決定する。

【０１０６】従来の第１の技術では、ｆｕｎｃＡ、ｆｕｎｃＢが同一キャッシュライン「青」を共有しており、それぞれの利用不可能集合には「青」が含まれておらず、従ってこれらｆｕｎｃＡ、ｆｕｎｃＢは直接の呼出関係がないため、関数呼出組み合わせ情報を元にした関数配置を行うと、これらの関数が同一キャッシュラインに乗ってしまう場合があり得、アプリケーションプログラムによっては、必ずしもこれら両関数間のキャッシュコンフリクトを削減することができなかった。これに対し、本実施の形態では、図９（Ｃ）に示す通り、ｆｕｎｃＡは「黄」、ｆｕｎｃＢは「青」にそれぞれ配置されるため、ｆｕｎｃＡとｆｕｎｃＢとが遷移を繰り返してもキャッシュコンフリクトが起きず、アプリケーションプログラムの実行スピードを向上することができる。

【０１０７】

【発明の効果】以上説明したように、本発明の命令キャッシュへの関数割付最適化装置、関数割付最適化方法及び関数割付最適化手順を記録した記録媒体は、関数呼出情報出力部と、関数呼出に応じた関数の遷移に対して該関数のＩＤ及び該関数の基本ブロックの順番の組合せを関数遷移毎に並べた関数基本ブロック遷移情報に出力する関数基本ブロック遷移情報出力部と、上記関数呼出組合せ情報を参照し関数呼出組合せ相互間で遷移回数を入替えた関数呼出回数入替データから成る呼出回数入替情報を生成し、次に生成した呼出回数入替情報を参照して関数をメモリ空間上のアドレスに仮配置した後、関数基本ブロック遷移情報を参照してキャッシュコンフリクトの回数を検出し、呼出回数入替データの中で、キャッシュコンフリクトの回数の最も少ないものに関数のメモリ配置を決定し対応する関数メモリ配置結果を出力する関数メモリ配置最適化部とを備えているので、以下の効果を奏する。

【０１０８】まず、第１の効果は、（１）ある関数の中で複数の関数が連続して呼ばれている場合、あるいは、（３）ループの中で呼ばれている場合等に対しても、キャッシュコンフリクトを削減するよう関数をメモリ空間上に配置することにより、アプリケーションプログラムの実行スピードを向上できることである。

【０１０９】その理由は、関数が連続して呼ばれている場合やループ中で呼ばれている等の呼出回数の多いもの

に着目して関数の呼出情報における呼出回数を入れ替え、それぞれの情報の重み付けを変更してから従来と同様の関数の配置処理を行なって仮配置をし、変更した中でもっともキャッシュコンフリクトの回数が少ないものを、関数の最終的なメモリ配置として決定するため、ある関数の中で複数の関数が連続して呼ばれている場合、あるいはループの中で呼ばれている場合などのようなアプリケーションプログラムの複雑さには依存しない処理であるからである。

【0110】また、第2の効果は、（3）ループ中で呼ばれる関数がまた別の関数を呼んでいる場合などに対しても、キャッシュコンフリクトを削減するよう関数をメモリ空間上に配置することにより、アプリケーションプログラムの実行スピードを向上できることである。

【0111】その理由は、関数の呼出情報における呼出回数を入れ替え、それぞれの情報の重み付けを変更してから従来の関数の配置処理を行なって仮配置をし、変更した中でもっともキャッシュコンフリクトの回数が少ないものを関数の最終的なメモリ配置を決定するため、ループ中で呼ばれる関数がまた別の関数を呼んでいる場合などのようなアプリケーションプログラムの複雑さには依存しない処理であるからである。

【図面の簡単な説明】

【図1】本発明の命令キャッシュへの関数割付最適化装置及びその処理手順の一実施の形態を示すブロック図である。

【図2】本実施の形態の関数呼出組合せ情報の一例を示す図である。

【図3】アプリケーションプログラムの一例を示す図である。

【図4】本実施の形態の関数基本ブロック遷移情報の一例を示す図である。

【図5】本実施の形態の命令キャッシュへの関数割付最適化装置の動作である関数割付最適化方法の一例を示すフローチャートである。

【図6】図5の呼出回数入替処理ステップの詳細処理を示すフローチャートである。

【図7】本実施の形態の呼出回数入替情報の一例を示す図である。

【図8】図5のキャッシュコンフリクト回数算出処理ステップの詳細処理を示すフローチャートである。

【図9】本実施の形態の関数メモリ配置結果を示す図である。

【図10】従来の命令キャッシュへの関数割付最適化装置及びその処理手順の一例を示すブロック図である。

【図11】従来の命令キャッシュへの関数割付最適化装置の動作である関数割付最適化方法の一例を示すフローチャートである。

【図12】従来の関数呼出グラフの構成を説明するための図である。

【図13】従来の関数配置とキャッシュラインの関係及び各関数の利用不可能集合の状態及び関数メモリ配置結果の一例を示す図である。

【符号の説明】

1　関数呼出情報出力部
2　関数基本ブロック遷移情報出力部
3，103　関数メモリ配置最適化部
4，104　関数メモリ配置結果
110　アプリケーションプログラム
111　関数呼出組合せ情報
112　関数基本ブロック遷移情報
113　呼出回数入替情報
120　関数呼出グラフ

【図2】

111

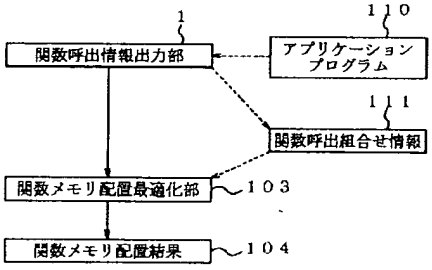
呼出元	呼出先	呼出回数
main	func	1
func	funcA	50
func	funcB	20
func	funcC	30
funcB	funcD	1
funcD	funcA	1

【図4】

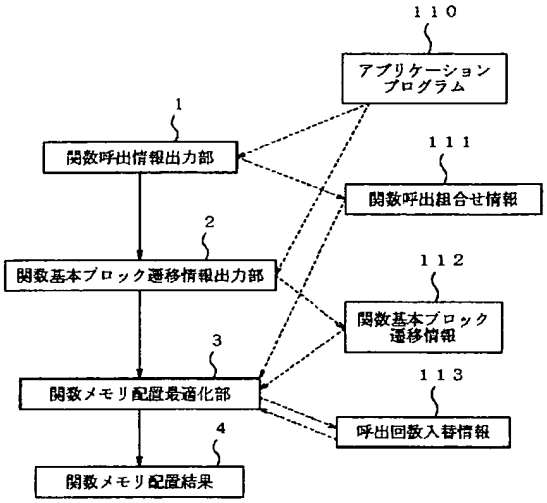
112

0－1，1－1，2－1，1－1，3－1，5－1，2－1，5－1，3－1
（以降同一配列を19回反復）
1－2，2－1，1－2，4－1
（以降同一配列を29回反復）
1－2，0－1

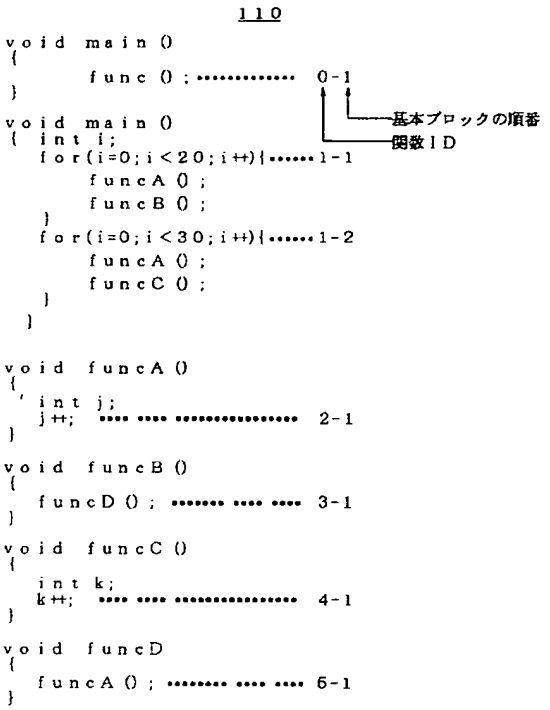
【図10】



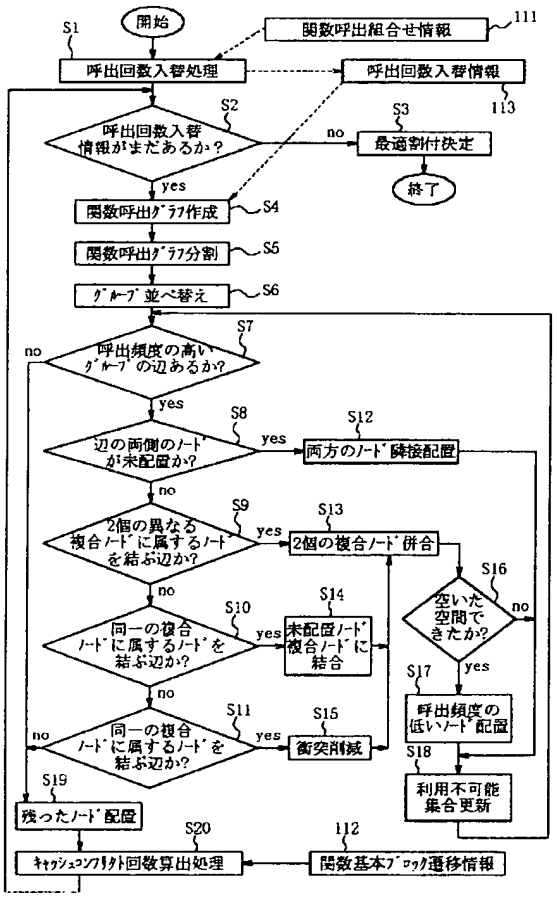
【図1】



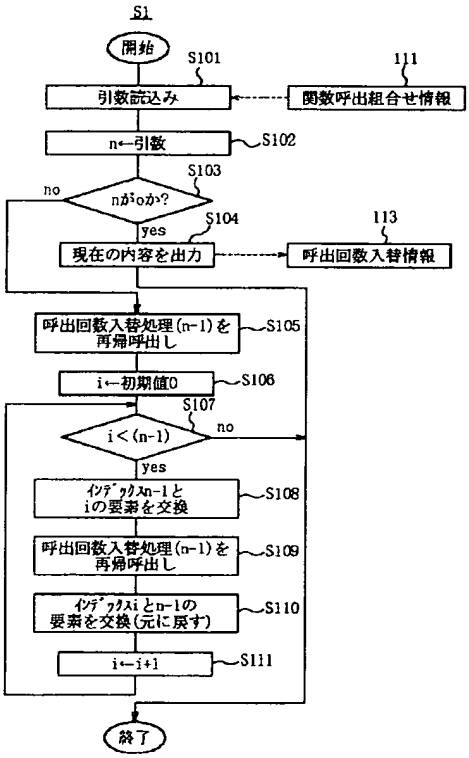
【図3】



【図5】



【図6】



【図7】

(A)

呼出元	呼出先	呼出回数
func	funcA	50
func	funcB	20
func	funcC	30

(B)

呼出元	呼出先	呼出回数
func	funcA	20
func	funcB	50
func	funcC	30

(C)

呼出元	呼出先	呼出回数
func	funcA	30
func	funcB	20
func	funcC	50

(D)

呼出元	呼出先	呼出回数
func	funcA	20
func	funcB	30
func	funcC	50

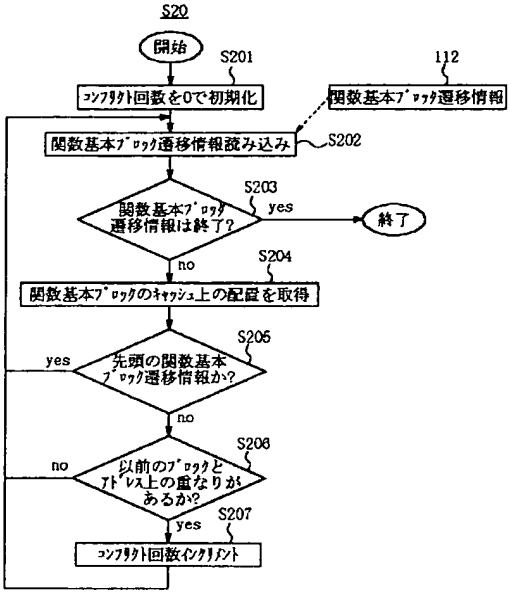
(E)

呼出元	呼出先	呼出回数
func	funcA	50
func	funcB	30
func	funcC	20

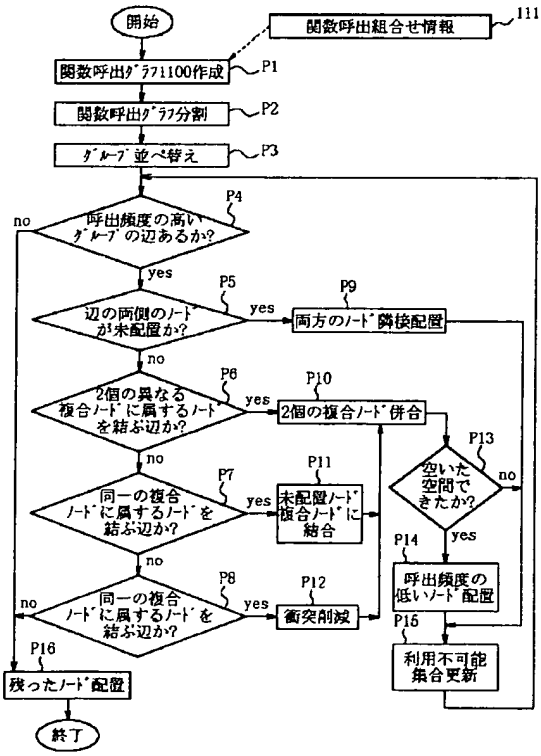
(F)

呼出元	呼出先	呼出回数
func	funcA	30
func	funcB	60
func	funcC	20

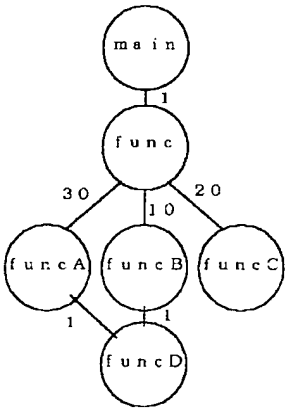
【図8】



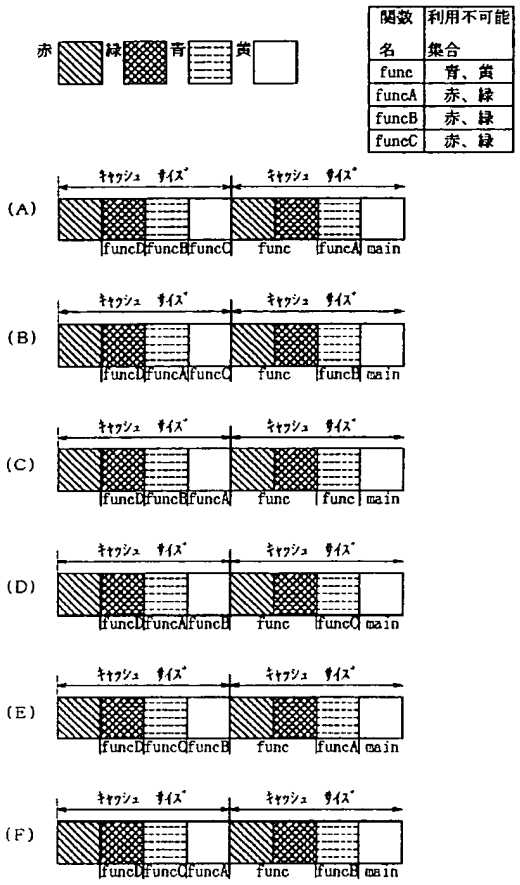
【図11】



【図12】

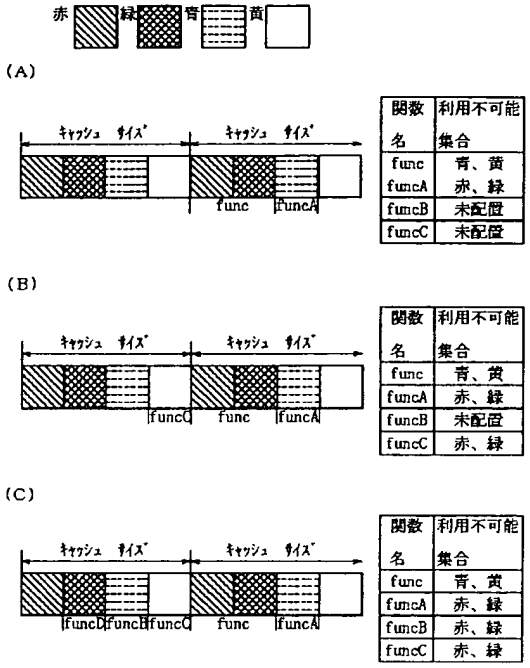


【図9】



関数名	利用不可能集合
func	青、黄
funcA	赤、緑
funcB	赤、緑
funcC	赤、緑

【図13】



(A)

関数名	利用不可能集合
func	青、黄
funcA	赤、緑
funcB	未配置
funcC	未配置

(B)

関数名	利用不可能集合
func	青、黄
funcA	赤、緑
funcB	未配置
funcC	赤、緑

(C)

関数名	利用不可能集合
func	青、黄
funcA	赤、緑
funcB	赤、緑
funcC	赤、緑